Learn About OpenEdge Database Docker Container Images

Table of Contents

Copyright

March 2020

Product version: Progress OpenEdge 12.2

Preface

Purpose

This content is an introduction to the OpenEdge database Docker container images. It describes the high-level tasks to perform when working with the database container images.

Audience

This content is for OpenEdge developers who want to test and validate changes to their business logic by creating OpenEdge databases on demand and dispose of them as soon as the purpose is served.

Organization

This content covers the following sections:

- Why use OpenEdge database Docker containers?
- What are the prerequisites for using OpenEdge database Docker containers?
- What are the main tasks to perform when using OpenEdge database Docker containers?

Documentation conventions

See Documentation Conventions for an explanation of the terminology, format, and typographical conventions used throughout the OpenEdge content library.

2

Why use an OpenEdge database in a Docker container

Reducing the time between the development of a feature and its use by customers is key for software organizations. Organizations need to take measures to reduce time invested in software delivery activities such as building, packaging, and testing. In the pre-virtualization days, building output from incremental changes to the codebase and then testing every newly built output required a dedicated infrastructure (networks, machines, relevant software installations and configurations, build and test scripts) and manual work. Organizations would either set up a clean infrastructure every time a new build was needed, or maintain the infrastructure exclusively for building and testing, which is an inefficient use of resources. Instead of spending time setting up or maintaining environments, you should be able to use resources when you need them and shut them down as soon as you finish with them.

Virtual machines addressed the problem to an extent. With a virtual machine, you can automate the entire process, which starts as soon as an increment is checked in to the codebase. If manually setting up the infrastructure took hours, then virtual machines brought the time down to minutes.

But, there are limitations to virtual machines. They cannot ensure that the building and testing of software components happen in real time. There is a considerable lag between the time developers check in their code changes and the time when the build process starts, for example. This is because starting a virtual machine and installing and configuring software applications take time. Though virtual machines ensure application-level isolation, they require a lot of resource overhead.

Container technology in general and Docker in particular are remarkable advances in this regard. With Docker, the infrastructure you need to build and test a software application can be set up on demand, in seconds, and can be removed just as easily. Docker ensures that there is isolation between containers without being resource intensive. Docker also enables application portability across machines and operating systems, which means that as long as you work with the same kernel, you can seamlessly deploy and run your application across different machines and different distributions of the operating system.

Because of its advantages, Docker is an important part of the continuous integration and continuous delivery or deployment (CI/CD) model. To enable Progress partners and customers to more efficiently implement their CI/CD pipelines, there is now support for deploying an OpenEdge database in a Docker container. For example, you can create and start a database in a Docker container on demand, run your tests, and then dispose of the container immediately after your tests are executed. As a result, you can focus on optimizing continuous testing of code increments without worrying about deployment details such as system resources, availability of a dedicated machine, and database setup.

Note: The OpenEdge database Docker container images are meant to be used in a development environment for building and testing purposes only. This is because containers are non-persistent. Consequently, when a container is deleted, all the data in it is lost. **DO NOT** use the database Docker container images in a production environment.

3

Prerequisites

Prerequisites for using OpenEdge database (12.1) Docker container images

To use an OpenEdge database (12.1) Docker container image, you must have:

- A Red Hat Enterprise Linux (7.5) machine
- Docker EE (18.09.x) installed on your Red Hat Enterprise Linux machine
- The Docker image of Oracle JDK (1.8.0.211-b12) on your machine
- An OpenEdge Enterprise RDBMS (12.1) or an OpenEdge Advanced Enterprise RDBMS (12.1) license and a 4GL development (12.1) license

Note: OpenEdge database (12.1) Docker images work with Progress Application Server (PAS) for OpenEdge (12.1).

Prerequisites for using OpenEdge database (12.2) Docker container images

To use an OpenEdge database (12.2) Docker container image, you must have:

- A Red Hat Enterprise Linux (7.5) machine
- Docker EE (18.09.x) installed on your Red Hat Enterprise Linux machine
- The Docker image of AdoptOpenJDK (11.0.4_11-jdk-hotspot) on your machine
- An OpenEdge Enterprise RDBMS (12.2) or an OpenEdge Advanced Enterprise RDBMS (12.2) license and a 4GL development (12.2) license

Note: OpenEdge database (12.2) Docker images work with Progress Application Server (PAS) for OpenEdge (12.2).

4

Work with OpenEdge database Docker container images: Main steps

To work with an OpenEdge database Docker container image:

1. Download the OpenEdge databse Docker container image to your machine.
2. Examine the contents of the `build` directory.
3. Modify the `config.properties` file as required.
4. Build a custom OpenEdge database Docker container image.
5. Run the custom image as a Docker container.
6. Test access to the OpenEdge database running inside the container.

5

Download an OpenEdge database Docker container image

You can download an OpenEdge database Docker container image from either of the following:

- Docker Hub
- Progress Download Center

Docker Hub

To download an OpenEdge database Docker image from the public Docker Hub repository:

1. Log into the Docker Hub from your Docker host.
2. Accept the terms and conditions.
3. Pull the image to your Docker host by running:
   - `docker pull store/progresssoftware/oedb:12.1.0_ent`, to download the OpenEdge Enterprise RDBMS (12.1) image
   - `docker pull store/progresssoftware/oedb:12.1.0_adv-ent`, to download the OpenEdge Advanced Enterprise RDBMS (12.1) image
   - `docker pull store/progresssoftware/oedb:12.2.0_ent`, to download the OpenEdge Enterprise RDBMS (12.2) image
   - `docker pull store/progresssoftware/oedb:12.2.0_adv-ent`, to download the OpenEdge Advanced Enterprise RDBMS (12.2) image

Progress Download Center

To download the required OpenEdge database Docker image from the Progress Download Center:

1. Log into the Progress Download Center.
2. Locate **PROGRESS_OE_DATABASE_CONTAINER_IMAGES_12.1.0_LNX_64.zip** or **PROGRESS_OE_DATABASE_CONTAINER_IMAGES_12.2.0_LNX_64.zip** using the **Product Search** link.
3. Download the ZIP file to your Docker host.
4. Extract the contents of the ZIP file to an appropriate location on the Docker host.

   The **PROGRESS_OE_DATABASE_CONTAINER_IMAGES_12.1.0_LNX_64.zip** file has the following contents:

   - The `build` directory, which has the configuration files, deployment scripts, and other artifacts
   - The `NOTICE.txt` file
   - The **PROGRESS_OE_ADVANCED_DATABASE_CONTAINER_IMAGE_12.1.0_LNX_64.tar.gz** archive, which is the base image of the OpenEdge Advanced Enterprise RDBMS
   - The **PROGRESS_OE_DATABASE_CONTAINER_IMAGE_12.1.0_LNX_64.tar.gz tar** archive, which is the base image of the OpenEdge Enterprise RDBMS

   The **PROGRESS_OE_DATABASE_CONTAINER_IMAGES_12.2.0_LNX_64.zip** file has the following contents:

   - The `build` directory, which has the configuration files, deployment scripts, and other artifacts
   - The `NOTICE.txt` file
   - The **PROGRESS_OE_ADVANCED_DATABASE_CONTAINER_IMAGE_12.2.0_LNX_64.tar.gz** archive, which is the base image of the OpenEdge Advanced Enterprise RDBMS
   - The **PROGRESS_OE_DATABASE_CONTAINER_IMAGE_12.2.0_LNX_64.tar.gz tar** archive, which is the base image of the OpenEdge Enterprise RDBMS

5. Run:

   - `docker load -i PROGRESS_OE_DATABASE_CONTAINER_IMAGE_12.1.0_LNX_64.tar.gz`, to load the OpenEdge Enterprise RDBMS (12.1) image
   - `docker load -i PROGRESS_OE_ADVANCED_DATABASE_CONTAINER_IMAGE_12.1.0_LNX_64.tar.gz`, to load the OpenEdge Advanced Enterprise RDBMS (12.1) image
   - `docker load -i PROGRESS_OE_DATABASE_CONTAINER_IMAGE_12.2.0_LNX_64.tar.gz`, to load the OpenEdge Enterprise RDBMS (12.2) image
   - `docker load -i PROGRESS_OE_ADVANCED_DATABASE_CONTAINER_IMAGE_12.2.0_LNX_64.tar.gz`, to load the OpenEdge Advanced Enterprise RDBMS (12.2) image

6

Examine the contents of the build directory

The `build` directory contains the following files and directories:

- `Dockerfile`—Contains the set of instructions used by the Docker daemon to build a Docker image. In addition to commands for setting various arguments and environment variables, the `Dockerfile` specifies, among other things, the base OpenEdge database and the JDK Docker images that are used to build the custom OpenEdge database Docker container image, and the various files and artifacts that must be copied into the custom image.

  Note: You must have the Docker image of a compatible JDK version loaded in your Docker host for the custom image to build successfully. This is because, starting with OpenEdge (12.1), JDK is no longer included in the OpenEdge installer.

- `config.properties`—Contains a number of properties that enable you to customize your database Docker container image.
- `build.sh`—Contains the instructions for building the custom database Docker container image, and validates the properties that are set in the `config.properties` file. The script will throw exceptions if the validation fails. If the validation succeeds, then the `build.sh` script uses the set properties to build the custom image.
- `hook-script.sh`—Contains sample code that you can use to enable the advanced features, such as Transparent Data Encryption, multi-tenancy, table partitioning, and Change Data Capture, on the database. If you want to use any of these features, you must enable them before starting your database.
- `artifacts`—Can have artifacts such as a database structure file, a data definition, database extents, or database backups.
- `abl-triggers`—If you have ABL triggers defined on your database, put them in this subdirectory. Any ABL program that must be run when a corresponding trigger event occurs must be in the `artifacts` subdirectory. For SQL triggers, there is no change: include the Java code within your SQL query, and run it against the database, as usual.
- `conf`—Contains the `startup.pf` parameter file.
- `license`—Must contain a valid OpenEdge Enterprise RDBMS (12.1/12.2) or Advanced Enterprise RDBMS (12.1/12.2) license, and a valid 4GL development (12.1/12.2) license.

7

Modify the config.properties file

You use the `DB_CREATE_METHOD` property to specify how you want your database to be created in your custom container image. There are four options to build your container image:

- If you want to create a database, then set the `DB_CREATE_METHOD` property to `customDB`. You must have either one of the following:
  - A database structure file (`.st`)
  - A data definition file (`.df`)
  - A `.df` file and database extents (`.d`)
  - A `.st`, a `.df`, and `.d` files in the `artifacts` subdirectory
- If you want a database restored from an existing database backup file, then set the `DB_CREATE_METHOD` property to `backupDB`. To use this method, you need a database backup file with the `.bck` extension in the `artifacts` subdirectory of the `build` directory. Incremental backups are not supported.
- If you want your database to be a copy of one of the sample databases that are shipped with the OpenEdge database, then set the:
  - `DB_CREATE_METHOD` property to `sampleDB`
  - `DB_NAME` property to the name you want to use for the database that will be created
  - `SAMPLE_DB_NAME` property to the name of the sample database that you want to use
- If you want your database to be a copy of one of your current databases that are in the Docker host filesystem, then:
  - Set the `DB_CREATE_METHOD` property to `externalDB`.
  - Specify the location of the existing database using the `EXTERNAL_DATABASE_PATH` property.

  Note: In order to use this method, you must have root user privileges.

In each of the four methods, you must provide:

- A custom name for the database to be created using the `DB_NAME` property.
- The name and tag of the base database Docker container image using the `DB_DOCKER_IMAGE_NAME` and the `DB_DOCKER_IMAGE_TAG` properties. The values you provide should correspond to one of the two base images.
- The name and tag of the custom image that you are building using the `NEW_DB_DOCKER_IMAGE_NAME` and the `NEW_DB_DOCKER_IMAGE_TAG` properties. The default values are `oedb1` and `latest`.
- The name and tag of the compatible JDK image using the `JDK_DOCKER_IMAGE_NAME` and the `JDK_DOCKER_IMAGE_TAG` properties.

Finally, you can change the JDK location inside the container to a different value by modifying the `JDK_DOCKER_IMAGE_JAVA_LOCATION` property value.

Note: To use OpenEdge database Docker container images, you must work with OpenEdge databases that use relative paths and that do not use any absolute paths.

8

Build a custom OpenEdge database Docker container image

To build a custom OpenEdge database Docker container image:

1. Navigate to the `build` directory.
2. Modify the `config.properties` file to set the necessary configuration properties for the custom image.
3. If you are using an OpenEdge Advanced Enterprise RDBMS license, enable the required advanced features using the `hook-script.sh` file.
4. Add the required artifacts (`.st`, `.df`, `.d`, or `.bck`) to the `artifacts` subdirectory.
5. If you want to use ABL triggers, add them to the `abl-triggers` subdirectory.
6. If you want to use database startup parameters, add them to the `/conf/startup.pf` parameter file.
7. Add a valid OpenEdge Enterprise RDBMS or OpenEdge Advanced Enterprise RDBMS license file and a valid 4GL development license file to the `license` subdirectory.
8. Run `sh build.sh`.

After the custom image successfully builds, run `docker image ls` to list all the Docker images in your Docker host. You should be able to see your newly built custom image in the list.

9

Run an OpenEdge database Docker container image

To run an OpenEdge database Docker container image, run the following command in your Docker host:

```
docker run -d -p <database_server_port>:<database_server_port> -p <database_minport>
-<database_maxport>:<database_minport>-<database_maxport> -e
DB_BROKER_PORT=<database_server_port>
 -e DB_MINPORT=<database_minport> -e DB_MAXPORT=<database_maxport>
<custom_image_name>
```

In the command:

- The `-d` parameter runs the Docker container in detached mode.
- The `-p` parameter publishes the database server port and the database minimum and maximum port range for interhost and intercotainer communication. If this parameter is not specified, you cannot access the OpenEdge database running inside a container from outside the container.
- The `-e` parameter sets an environment variable for the container.

10

Test access to the OpenEdge database running inside a container

A simple way to test whether the OpenEdge database running inside a Docker container can be accessed from outside the container is to use an ABL client such as Data Administration to establish a connection with the database.

To establish a connection with the OpenEdge database running inside a container using Data Administration:

1. Launch **Data Administration**.
2. Select **Database > Connect** in the menubar.
3. Enter the database name in the **Physical Name** field.
4. Click **Options**.
5. Enter the IP address of the Docker host in the **Host Name** field.
6. Enter the database server port in the **Service Name** field.
7. If user credentials are required to access the database, then enter a valid user ID and password in the **User ID** and **Password** fields.
8. Click **OK**.

   You should be able to connect to the OpenEdge database in a container.